Nonlinear Programming

A logarithmic barrier cutting plane method for convex programming*

D. den Hertog, J. Kaliski, C. Roos and T. Terlaky☆
Faculty of Technical Mathematics and Informatics,
Delft University of Technology, Delft, The Netherlands

The paper presents a logarithmic barrier cutting plane algorithm for convex (possibly non-smooth, semi-infinite) programming. Most cutting plane methods, like that of Kelley, and Cheney and Goldstein, solve a linear approximation (localization) of the problem and then generate an additional cut to remove the linear program's optimal point. Other methods, like the "central cutting" plane methods of Elzinga–Moore and Goffin–Vial, calculate a center of the linear approximation and then adjust the level of the objective, or separate the current center from the feasible set. In contrast to these existing techniques, we develop a method which does not solve the linear relaxations to optimality, but rather stays in the interior of the feasible set. The iterates follow the central path of a linear relaxation, until the current iterate either leaves the feasible set or is too close to the boundary. When this occurs, a new cut is generated and the algorithm iterates. We use the tools developed by den Hertog, Roos and Terlaky to analyze the effect of adding and deleting constraints in long-step logarithmic barrier methods for linear programming. Finally, implementation issues and computational results are presented. The test problems come from the class of numerically difficult convex geometric and semi-infinite programming problems.

Keywords: Column generation, convex programming, cutting plane methods, decomposition, interior point method, linear programming, logarithmic barrier function, nonsmooth optimization, semi-infinite programming.

1. Introduction

In this paper, we consider the following general (nonsmooth) convex programming problem:

$$\begin{aligned} &\text{minimize} && f_0(y) \\ &\text{subject to} && y \in \mathcal{F}, \end{aligned} \tag{1}$$

where

$$\mathcal{F} := \{y \in \mathbb{R}^m : f_i(y) \le 0, \quad 1 \le i \le n\}$$

and the functions $f_i(y)$, $0 \le i \le n$, are assumed to be convex. The above convex programming problem may be restated with a linear objective as follows:

* This work wass completed under the support of a research grant of SHELL.
☆ On leave from the Eötvös University, Budapest, and partially supported by OTKA No. 2116.

$$\begin{aligned} &\text{maximize} && \eta \\ &\text{subject to} && f_0(y) + \eta \le 0, \\ &&& f_i(y) \le 0, \quad 1 \le i \le n. \end{aligned}$$

Therefore, without loss of generality we may assume that $f_0(y)$ is linear in (1). Taking $f_0(y) = -b^T y$, we will use the following format.

(CP)
$$\begin{aligned} &\text{maximize} && b^T y \\ &\text{subject to} && f_i(y) \le 0, \quad 1 \le i \le n, \end{aligned}$$

where $b \in \mathbb{R}^m$. We will assume that $\|b\| = 1$ and that $\mathcal{F}$ is compact.[1] Further, we assume that $\mathcal{F}^0$, the interior of $\mathcal{F}$, is not empty. This condition is equivalent to the Slater condition used by Elzinga and Moore [4].

The first and most popular cutting plane algorithm for convex programs was independently derived by Kelley [19] and Cheney and Goldstein [2]. Since then, several other cutting plane methods for convex programming have been developed. One of the advantages of cutting plane methods is that the subproblems generated are linear programming (LP) problems. Since these successive LP problems differ only slightly from the previous subproblems, efficient "warm start" techniques are applicable. Furthermore, no line searches are necessary, and the near linearity of nonlinear functions is utilized. A potential drawback with these methods is that the size of the generated LP subproblem may increase to become very large. Although this growth can be handled by efficient constraint dropping strategies, the generation of a large number of cutting planes might still be necessary. Another serious drawback of Kelley's method is that no feasible point is generated by the subproblems; it produces a sequence of points which are optimal in terms of the LP relaxations but are infeasible (except for the final solution) with respect to the original convex problem. Contrary to these methods, our algorithm always remains in the interior of the feasible set.

The central cutting plane methods of Elzinga and Moore [4, 10] and Goffin and Vial [9] are considered to be efficient. Also see Kortanek and No [22], Haurie et al. [7], Bahn et al. [1], Mitchell and Todd [24], and Kaliski and Ye [18]. Given the current linear localization, the method of Elzinga and Moore calculates the center of the largest inscribed hypersphere. If the center is feasible, then they add an objective cut; if the center is infeasible, then a new separating hyperplane is generated. Goffin and Vial choose a similar approach: instead of the center of the largest inscribed sphere, they use the analytic center (the point which maximizes the dual logarithmic barrier function, see Sonnevend [27]) of the actual linear localization. These two methods generate both feasible and infeasible points during the algorithm.

[1] We use the compactness assumption just for the sake of simplicity. In fact, all of our results remain valid if we only assume that the level sets of $b^T y$ in $\mathcal{F}$ are bounded.

Convex programming problems are frequently solved as semi-infinite (convex) programming problems. Any convex set can be constructed as the intersection of (a possibly infinite number of) halfspaces, where the boundary of these halfspaces are supporting hyperplanes of the feasible set. An overview of methods for semi-infinite programming can be found in the survey of Hettich and Kortanek [12].

Contrary to the existing techniques, we develop a method which does not solve the linear relaxations to optimality, but rather stays in the interior of the feasible set. The iterates follow the central path of a linear relaxation, until the current iterate either leaves the feasible set or is too close to the boundary. When this occurs, a new cut is generated and the algorithm iterates. We use the tools developed by den Hertog et al. [15,16] to analyze the effect of adding and deleting constraints in long-step logarithmic barrier methods for linear programming. In this way, we combine interior point methods (IPM) with a new cutting plane (decomposition) method. Since a convex programming problem can be considered as a semi-infinite programming problem, our algorithm can also be used as an IPM technique to find solutions for semi-infinite programming problems.

In [15], a build-up strategy for the long-step logarithmic barrier method is presented. In [16], a logarithmic barrier method for LP is studied which allows adding and deleting constraints. Starting with a (small) subset of the constraints, this algorithm follows the corresponding central path of the subproblem until the iterate is close to (or violates) one of the other constraints. Given some proximity parameter t_a, new constraints are added to the system whenever their dual slack values fall below t_a. This process is repeated until the iterate is close to optimality. If $-\log_2 t_a < O(L)$, then this build-up algorithm finds a solution with duality gap less than 2^{-2L} within $O(q^*L)$ Newton iterations, where L is the bit length of the data and q^* is the number of constraints in the final system. In [16], also a constraint dropping strategy is presented which maintains this worst-case complexity. We use these results in our algorithm.

The paper is organized as follows. In section 2, we review the results of [15,16] corresponding to adding and deleting constraints in logarithmic barrier methods. In section 3, our logarithmic barrier algorithm for convex programming problems is discussed. In section 4, implementation strategies and computational results for this new method are presented. Finally, some conclusions are made.

Notation

The following notations are used throughout this paper. e denotes the vector of all ones. Given an $m \times n$ matrix A, its columns are denoted by a_i, $i = 1,\ldots,n$. Given an n-dimensional vector s, S denotes the $n \times n$ diagonal matrix whose diagonal entries are the coordinates s_j of s; s^T is the transpose of the vector (matrix) s. Finally, $\|s\|$ denotes the l_2 norm of s.

2. Adding and deleting constraints in logarithmic barrier IPMs for LP

In this section, we review some of the known results for logarithmic barrier methods applied to linear programming. After a brief introduction, we highlight the effects of adding and deleting constraints and present the build-up and down algorithm of [16].

2.1. A LONG-STEP PATH-FOLLOWING METHOD

Consider the dual linear programming problem:

$$\textbf{(D)} \qquad \begin{aligned} &\text{maximize} && b^{\mathrm{T}}y \\ &\text{subject to} && A^{\mathrm{T}}y + s = c, \quad s \geq 0. \end{aligned}$$

A is an $m \times n$ matrix, b and c are m- and n-dimensional vectors, respectively; the m-dimensional vector y is the variable in which the maximization is done, and s is the n-dimensional dual slack vector. As usual, L denotes the bit length of the input data of **(D)**. We also make the standard assumption that A has full row rank and that the feasible set of **(D)** has bounded level sets and a nonempty interior.

We consider the dual logarithmic barrier function

$$f(y, \mu) := \frac{b^{\mathrm{T}}y}{\mu} + \sum_{j=1}^{n} \ln s_j, \tag{2}$$

where μ is a positive parameter. f achieves a maximum value at a unique interior point $y(\mu)$ [23]. We define the related slack as $s(\mu) = c - A^{\mathrm{T}}y(\mu)$. The central path of **(D)** is defined as the set of solutions $y(\mu)$ for $\mu > 0$.

Roos and Vial [26] introduced the following measure for the distance of an interior feasible point to the central point $y(\mu)$:

$$\delta(y, \mu) := \min_{x} \left\{ \left\| \frac{Sx}{\mu} - e \right\| : Ax = b \right\}. \tag{3}$$

The unique solution of the minimization problem in the definition of $\delta(y, \mu)$ is denoted by $x(y, \mu)$. If the Newton direction for (2) is denoted by p, then it can be verified that $\delta(y, \mu) = \|S^{-1}A^{\mathrm{T}}p\|$ [13]. A closed formula for p is

$$p = (AS^{-2}A^{\mathrm{T}})^{-1}\left(\frac{b}{\mu} - AS^{-1}e\right), \tag{4}$$

and for $x(y, \mu)$

$$x(y,\mu) = \mu S^{-1}e + \mu s - 2A^{\mathrm{T}}p$$
$$= \mu S^{-1}e + \mu s - 2A^{\mathrm{T}}(AS^{-2}A^{\mathrm{T}})^{-1}\left(\frac{b}{\mu} - AS^{-1}e\right). \tag{5}$$

It can easily be verified that if y is feasible, then $y = y(\mu) \Leftrightarrow d(y,\mu) = 0$.

Briefly stated, long-step logarithmic barrier methods proceed as follows. Starting from an interior feasible point, the method generates the Newton direction (3) and searches along this direction to maximize (1). Once the maximum point along the Newton direction is found, the iterate is updated and a new Newton direction is generated. The process continues until the iterate gets close to the current center ($\delta(y,\mu) < 1/2$). Then the barrier parameter is reduced by a fixed fraction $1-\theta$, and the entire process continues. Note that all the iterates stay feasible, since the logarithmic barrier function assumes its minimum in the interior of the feasible set.

Suppose the logarithmic barrier method, as defined in [14], starts with barrier parameter $\mu_0 > 0$ and $0 < \theta < 1$, independent of n (say $\theta = \frac{1}{2}$). Then after at most $O(-\log_2 t_c + \ln n\mu_0)$ reductions of the barrier parameter, the long-step algorithm ends up with both a primal and a dual feasible solution such that $x^{\mathrm{T}}s \le t_c$. Each reduction of the barrier parameter requires at most $O(n)$ Newton iterations. See [14].

2.2. SHIFTING, ADDING AND DELETING CONSTRAINTS

The following results pertaining to the shifting, adding and deleting of constraints in logarithmic barrier methods are proved in [15,16]. We note that Ye [29] has also proved some of these results.

The full index set $\{1,\ldots,n\}$ is denoted by N and $Q \subseteq N$. In this section, we use the notation $(AS^{-2}A^{\mathrm{T}})_Q$ to denote $AS^{-2}A^{\mathrm{T}}$ restricted to the columns of A in the index set Q, i.e.

$$(AS^{-2}A^{\mathrm{T}})_Q := \sum_{i\in Q} \frac{a_i a_i^{\mathrm{T}}}{s_i^2}.$$

Moreover, we define

$$\|x\|_Q := \sqrt{x^{\mathrm{T}}(AS^{-2}A^{\mathrm{T}})_Q^{-1}x}.$$

Suppose the first constraint is shifted by a fraction of the current slack s_1, i.e. the constraint

$$a_1^{\mathrm{T}}y \le c_1$$

is replaced by

$$a_1^{\mathrm{T}}y \le c_1 - \varepsilon s_1, \quad 0 \le \varepsilon \le 1.$$

Let the superscript * refer to this new situation; so $s_1^* = (1-e)s_1$ and $s_i^* = s_i$ for $i = 2,\ldots,n$. The following lemma shows the effect of shifting.

LEMMA 1

If we shift the first constraint by $\varepsilon s_1(\mu)$, then

$$\delta^*(y,\mu) \le \delta(y,\mu) + \varepsilon(1+\delta(y,\mu)),$$
$$(1-\varepsilon)s_1(\mu) \le s_1^*(\mu) \le s_1(\mu),$$
$$f^*(y^*(\mu),\mu) \le f(y(\mu),\mu) - \varepsilon.$$

□

Suppose we add a constraint, $a_0^T y \le c_0$ say. Let $s_0 > 0$ be the corresponding slack variable and define

$$\delta_0 = \frac{s_0}{\|a_0\|_N}.$$

Let the superscript * refer to this new situation. The next lemma analyzes the effect of adding a constraint on δ.

LEMMA 2

One has

$$\delta^*(y,\mu) \le \begin{cases} \dfrac{\delta_0\delta(y,\mu)+1}{\sqrt{\delta_0^2+1}} & \text{if } \delta_0 \ge \delta(y,\mu), \\ \sqrt{1+\delta(y,\mu)^2} & \text{if } \delta_0 < \delta(y,\mu), \end{cases}$$

and

$$s_0^*(\mu) \ge s_0(\mu).$$

Furthermore, if $\delta(y,\mu) \le \frac{1}{4}$, then

$$f^*(y^*(\mu),\mu) - f^*(y,\mu) \le \tfrac{1}{3} + \max\left(0, \ln\frac{4}{\delta_0}\right).$$

□

Suppose the first constraint $a_1^T y \le c_1$ is removed and assume that the remaining constraint matrix still has full rank. Let the superscript * refer to this new situation. Now, defining $N^* := N \setminus \{1\}$ and

$$\delta_1 = \frac{s_1}{\|a_1\|_N},$$

we have the following result:

LEMMA 3

$$\delta^*(y,\mu) \le \delta(y,\mu) + \frac{1+\delta(y,\mu)}{\delta_1}.$$

□

2.3. BUILD-UP AND DOWN STRATEGIES

Based on the analysis presented in the previous section, build-up (adding constraints) and build-down (deleting constraints) strategies are given in [16]. As described below, an algorithm based on these strategies can be developed which solves the original problem while only operating on subsets of N.

For convenience, we assume the problem contains box constraints ($-e \leq y \leq e$) whose index set will be denoted as J ($J \subseteq N$). Starting with an interior feasible iterate (with respect to the original problem), a small subset $\overline{Q} \subseteq N \backslash J$ is chosen. Let $Q = \overline{Q} \cup J$. Once Q has been chosen, the algorithm proceeds with respect to the subset dual problem D_Q.

$$(\mathbf{D}_Q) \qquad \max \{b^T y : a_i^T y \leq c_i, i \in Q\}.$$

However, we check in each iteration if there is an index $i \notin Q$ such that

$$s_i < t_a \quad \text{or} \quad s_i < t_a \hat{s}_i, \tag{6}$$

where t_a is some "adding" parameter, and $\hat{s}$ is the slack vector of the dual iterate which was almost centered ($\delta(\hat{y}, \hat{\mu}) < 1/4$) with respect to the previous value of the barrier parameter. If there is such a constraint, we add it to our system, go back to the previous iterate (for which $s_i \geq t_a$ and $s_i \geq t_a \hat{s}_i$) and continue the logarithmic barrier process. Consequently, all iterates remain feasible for the original problem. This is the build-up strategy.

When the iterate is close to the central path, the slack values associated with the current iterate are checked. If there is an i such that $s_i \geq t_d$, where t_d is a "deleting" parameter, we remove it from our current system, since it is likely that this constraint will be nonbinding in an optimal solution. After removing constraints, we recenter as necessary. A deleted constraint may indeed be binding for the optimal solution, but this causes no problems because the constraint will then be added in subsequent iterations. Cycling (the infinite adding and deleting of a set of constraints) is avoided by only deleting constraints when the iterate is close to the central path.

Before describing the algorithm, we introduce some notations. Let $\delta_Q(y, \mu)$, $f_Q(y, \mu)$ and p_Q denote the δ-measure, the barrier function, and the Newton direction, respectively, with respect to the subsystem Q. The algorithm is as follows.

BUILD-UP AND DOWN ALGORITHM

Input:

$\mu = \mu_0$ is the barrier parameter value;

t_c is a convergence parameter;

θ is the reduction parameter, $0 < \theta < 1$;

Q is the initial subset of constraints;

y is a given interior feasible point for (**D**) such that $\delta_Q(y, \mu) \leq \frac{1}{4}$ (see e.g. [25]);

Output:
x_Q is the primal solution vector;
(y, s) is the dual solution vector/slasck for (**D**);

begin
  while $s^T x(y, \mu) > t_c$ **do**
    begin
      Delete-Constraints;
      $\mu := (1 - \theta)\mu$;
    end
    Calculate x_Q using (5);
end.

PROCEDURE CENTER-AND-ADD-CONSTRAINTS

Input:
t_a is an "adding" parameter;
Q the set of the currently active constraints;
y is the current iterate;

Output:
Q the set of the currently active constraints;
$\hat{y}$ is a centered solution;

begin
  while $\delta_Q(y, \mu) > \frac{1}{4}$ **do**
    begin
      $\tilde{y} := y$;
      $\tilde{\alpha} := \arg\max_{\alpha > 0}\{f_Q(y + \alpha p_Q, \mu) : s_i - \alpha a_i^T p_Q > 0, \forall i \in Q\}$;
      $y := y + \tilde{\alpha} p_Q$;
      if $\exists i \notin Q : s_i < t_a \max(1, \hat{s}_i)$ **then**
      begin
        $y := \tilde{y}$;
        $Q := Q \cup \{i : s_i < t_a \max(1, \hat{s}_i), i \notin Q\}$;
      end
    end
  $\hat{y} := y$
end.

PROCEDURE DELETE-CONSTRAINTS

Input:
$t_d \geq 4$ is a "deleting" parameter;

Q the set of the currently active constraints;
$\hat{y}$ is a centered solution;

Output:
Q the set of the currently active constraints;
$\hat{y}$ is a centered solution;

begin
  for $i := 1$ to n **do**
    if $i \in Q \setminus J$ **and** $s_i \geq t_d$ **then**
    begin
      $Q := Q \setminus \{i\}$;
      if $\delta_Q(y, \mu) \geq \frac{1}{4}$ **then** Center-and-Add;
    end
end.

Remarks

(1) The selection of the parameters t_a, t_d, t_c is adaptive. The choice of the parameters of course affects both the practical performance and the theoretical complexity of the algorithms. How the theoretical complexity is influenced by the parameters is presented in the following subsection. The choice of these parameters in our implementation is discussed in section 4.

(2) The termination criteria $s^T x(y, \mu) > t_c$ says that the algorithm stops if the current duality gap is smaller than t_c. The dual solution (y, s) is dual feasible. The primal solution $x_Q(y, \mu)$ is feasible and centered for the problem $\{A_Q x_Q = b, x_Q \geq 0\}$. Defining $x_i = 0$ if $i \notin Q$ results in a primal feasible solution to the original (primal) problem.

2.4. COMPLEXITY OF THE ALGORITHM

The following complexity results are proved in [16]. Let q be the cardinality of the current subset Q, and let θ be independent of q (say $\theta = \frac{1}{2}$).

THEOREM 1

Between two reductions of the barrier parameter, the Build-Up and Down Algorithm requires at most $O(q + r(-\log_2 t_a) + r \ln r)$ Newton iterations if r constraints were added. □

Lemma 3 is useful in analyzing the Delete-Constraints procedure. If the ith constraint is deleted and the iterate is near the μ-center (i.e. $\delta_Q(y, \mu) \leq 1/4$) whereas $s_i \geq t_d \geq 4$, then

$$\delta^*(y,\mu) \leq \frac{1}{4} + \frac{1+\frac{1}{4}}{4}\frac{1}{\sqrt{2}} < \frac{1}{2}.$$

As a consequence of lemma 1, we have that recentering costs at most $O(1 + r(-\log_2 t_a) + r\ln r)$ Newton iterations, if r constraints have to be added.

A constraint which is removed during the outer iterations may be added during each inner loop. It is easy to develop strategies to prevent this; e.g. a constraint may be deleted only once (or a fixed number of times). For this purpose, we introduce K as the maximal number of times a constraint may be deleted. Clearly, K will be not larger than the number of updates of μ; i.e. $K \leq (-\log_2 t_c) + \ln q^* \mu^0 := K_{\max}$, where q^* denotes the maximum number of constraints at any time included in the subsystem at any time.

THEOREM 2

After at most $O((K+1)(q^* \ln q^* - q^* \log_2 t_a) + q^*(\ln q^* \mu_0 - \log_2 t_c))$ Newton iterations, a t_c-optimal solution has been found for (**D**), where q^* denotes the maximum number of constraints in the subsystem during the whole process. □

COROLLARY

If $K = O(1)$ and $-\log_2 t_c = O(L)$, then the Build-Up and Down Algorithm converges in $O(q^* L)$ Newton iterations.

If $K = K_{\max}$ and $-\log_2 t_a = O(\ln q)$, where q is the current number of constraints in our subsystem, then the Build-Up and Down Algorithm converges in $O(q^* L \ln q^*)$ Newton iterations. □

3. A path-following cutting plane method for convex programming

Now we present our algorithm for convex programming problems. Recall that the problem (**CP**) to be solved is stated as follows:

$$\begin{aligned} (\mathbf{CP}) \qquad & \text{maximize} \quad b^T y \\ & \text{subject to} \quad f_i(y) \leq 0, \quad 1 \leq i \leq n. \end{aligned}$$

As in the previous section, we assume that the problem contains box constraints $(-e \leq y \leq e)$ whose index set will be denoted as J.

Our algorithm is a straightforward application of the Build-Up and Down Algorithm of the previous section. The algorithm below can be used to find a primal solution to convex, nonsmooth and semi-infinite programming problems.

In the algorithm, we generate a sequence of linear programming relaxations of the original convex programming (**CP**) problem. Starting from a point in the

interior of the feasible region of (**CP**), a new potential iterate is generated as follows. With respect to the LP relaxation, a Newton step is calculated, producing an ascent direction for the dual logarithmic barrier function. Along this direction, a line search is performed to find the minimum of the barrier function in the interior of the feasible set. To avoid the relaxation boundary, we choose the point which is 90% of the distance to the minimum point along the Newton direction. If the new iterate violates or is too "close" to the boundary of a convex constraint, then the previous iterate is retained and a new supporting hyperplane is added to improve the approximation; otherwise, the new iterate is accepted. If the current iterate is centered, a deletion rule is used to eliminate redundant constraints, which keeps the size of the LP relaxation as small as possible.

Formally, the algorithm is stated as follows.

A PATH-FOLLOWING CUTTING PLANE ALGORITHM

Input:

$\mathcal{P}$ is a convex polytope (the initial LP relaxation) such that $\mathcal{F} \subset \mathcal{P}$;
$\mu = \mu_0$ is the barrier parameter value;
t_c is a convergence parameter, $0 < t_c$;
θ is the μ reduction parameter, $0 < \theta < 1$;
Q is the initial index set of the (linear) constraints of $\mathcal{P}$;
$y \in \mathcal{F}^0$ is a given interior feasible point such that $\delta_Q(y, \mu) \leq \frac{1}{4}$;

begin
  while $s^T x(y, \mu) > t_c$ **do**
    begin
      Delete-Constraints;
      $\mu := (1 - \theta)\mu$;
      Center-and-Add-Constraints;
    end
end.

PROCEDURE CENTER-AND-ADD-CONSTRAINTS

Input:

t_a is an "adding" parameter, $0 < t_a$;
Q the set of the currently active cuts;
y is the current iterate;

Output:

Q the set of the currently active cuts;
$\hat{y}$ is a centered solution;

begin
  while $\delta_Q(y,\mu) > \frac{1}{4}$ **do**
    begin
      $\tilde{y} := y$;
      $\tilde{\alpha} := \arg\max_{\alpha>0}\{f_Q(y+\alpha p_Q,\mu) : s_i - \alpha a_i^T p_Q > 0, \forall i \in Q\}$;
      $y := y + 0.9\tilde{\alpha} p_Q$;
      if $\exists k : -f_k(y) < t_a$ **then**
      begin
        Add-Cut;
        $y = \tilde{y}$;
      end
    end
  $\hat{y} := y$;
end.

PROCEDURE DELETE-CONSTRAINTS

Input:
$t_d \geq 4$ is a "deleting" parameter;
Q the set of indices of the currently active cuts;
$\hat{y}$ is a centered solution;

Output:
Q the set of indices of the currently active cuts;
$\hat{y}$ is a centered solution;

begin
  for $i := 1$ to n **do**
    if $i \in Q \setminus J$ **and** $s_i \geq t_d$ **then**
    begin
      $Q := Q \setminus \{i\}$;
      if $\delta_Q(y,\mu) \geq \frac{1}{4}$ **then** Center-and-Add-Constraints;
    end
end.

PROCEDURE ADD-CUT

Input:
$\tilde{y} \in \mathcal{F}$ and $y \notin \mathcal{F}$ or y is close to the boundary of $\mathcal{F}$;
$\overline{Q}$ the set of indices of the currently active cuts;

Output:
Q the set of indices of the currently active cuts;

begin
  Call ORACLE$(\tilde{y}, y(a^T, \gamma))$
  $Q = \overline{Q} \cup \{(a^T, \gamma)\}$
end.

The ORACLE routine provides a cut in the case that the new iterate y is either outside the convex feasible region or is too near its boundary. The nature of the ORACLE changes depending on the characteristics of the convex region. If $\mathcal{F}$ is differentiable and its gradients can be calculated with little effort, the ORACLE will return the gradient; if $\mathcal{F}$ is nonsmooth, the ORACLE returns a subgradient; if function evaluations are expensive, the ORACLE is constructed to minimize line searching. Below, we give a concrete example for when the functions f_k are differentiable and the function evaluations are cheap. This is the case for the problem set solved later in this paper.

ORACLE CUT (Smooth functions, cheap evaluation)

Input:
$\tilde{y} \in \mathcal{F}$ and $y \notin \mathcal{F}$ or y is close to the boundary of $\mathcal{F}$;

Output:
cut $a^T y \leq \gamma$;

begin
  If $y \notin \mathcal{F}$ **then**
    Find y^b as the boundary point of $\mathcal{F}$ on the line segment $(\tilde{y}, y)$;
  Else
    Find y^b as the boundary point of $\mathcal{F}$ on the line segment $\tilde{y} + \alpha(y - \tilde{y})$ where $\alpha \geq 0$;
  Let f_k be a constraint with $f_k(y^b) = 0$;
  $a = \nabla f_k(y^b)$, $\gamma = a^T y^b$;
  $\mathcal{P} = \mathcal{P} \cap \{a^T y \leq \gamma\}$;
end.

Note that y^b as defined in the ORACLE above is found through standard line-search techniques. If it is difficult to find the boundary point for the above ORACLE (due to numeric problems or expensive function evaluations), then a cut can be generated based on y or $\tilde{y}$ (without line searching) as follows:

$$\begin{aligned} a &= \nabla f_k(y), \\ \gamma &= a^T y - f_k(y), \end{aligned} \tag{7}$$

or

$$a = \nabla f_k(\tilde{y}),$$
$$\gamma = a^T \tilde{y} - f_k(\tilde{y}).$$

To demonstrate the validity and viability of these alternate techniques, we also report computational results using (7) as a cut generation technique. See section 4.3.

3.1. ABOUT CONVERGENCE

A straightforward convergence proof of the above algorithm can be obtained by using the results of the discretization of semi-infinite programming problems (see e.g. Gustafson [11]). Since any convex set can be presented as an intersection of an infinite number of halfspaces, convex programming problems can always be interpreted as semi-infinite programming problems. Gustafson [11] proves the following theorem.

THEOREM 3 (Gustafson [11])[2)]

Given a semi-infinite programming problem

$$\max\{b^T y \mid a(v)^T y \leq \gamma(v), v \in C\},$$

where C is a compact set and the functions $a(v)$ and $\gamma(v)$ are continuous on the set C. Then there is a finite subset $T \subset C$ such that the semi-infinite programming problem is computationally equivalent to the linear program induced by the finite set T.

"Computationally equivalent" above means that the two problems are equivalent up to a certain given precision. Of course, the finite discretization of the semi-infinite programming problem (or equivalently the finite discretization of the convex programming problem) can be very large. Nevertheless, theoretically it is enough to solve a finite discretization (a finite LP problem) of the original problem. For the induced discretization, the results of the Build-Up and Down Algorithm of the previous section apply; starting with a small subset of constraints, constraints are added at the closest discretization point to the boundary point in question. This proves the practical convergence of the algorithm by solving a computationally equivalent problem. Recall the complexity of the algorithm depends on the number of the generated cuts, which can be enormous (possibly exponential in the original problem dimensions). As a consequence, proving the polynomial complexity of our algorithm remains an open question – a subject of further research.

[2)] Remember the assumption (the feasible set is compact) and the remark made about its relaxation (it is sufficient to assume that the level sets are bounded) in section 1. It is clear from his development that Gustafson really meant to include an additional assumption like bounded level sets.

3.2. COMPARISON TO OTHER CUTTING PLANE METHODS

Kelley's cutting plane method [19] solves an LP approximation of the problem (**CP**) in each step. LP problems are traditionally solved by the simplex method. The solutions to these localizations are all infeasible for (**CP**). This has two negative consequences. The first is that the method is unstable since the subsequent solutions may be located at vertices very far from each other. The second is that if the algorithm is terminated before optimality, no feasible solution is known, therefore wasting the computational effort.

Elzinga and Moore's [4] central cutting plane method eliminates these disadvantages. Centering (calculating the ball center) ensures some stability and hopefully at least some of the iterates are feasible; when stopped prematurely, the algorithm still produces some useful information. Kortanek and No [22] report some encouraging computational results.

The central cutting plane method of Goffin and Vial [9] also enjoys these same advantages. (The relation of the two methods was discussed in section 1.) In this method, the analytic center of the polytope is calculated by using a projective interior point algorithm. All efficient interior point methods follow the central path of the LP problem, which indicates that this method might even be more efficient than the cutting plane method of Elzinga and Moore. Impressive computational results are reported in [1].

Our method shares the advantages of the above central cutting plane methods. By following the central path of the actual LP relaxations, our method maintains a centering component which provides a stability similar to that mentioned above. By adding new cuts, the actual center moves, but in the controlled way as described in lemmas 1 and 2. The same controlled movement of the central path occurs when a loose cut is deleted.

The main advantage of our approach is that in the above central cutting plane methods the LP relaxation is fixed while the new center is calculated. We dynamically refine the LP approximation of (**CP**) as the iterates draw close to the central path. In this way, significant computational savings are gained. Another advantage of our approach is that we start from a feasible point of the convex programming problem and feasibility is always preserved. Therefore, we can stop at any time. Moreover, when an iterate is centered, there is a bound for the duality gap of the LP problem which in turn also provides a bound for the solution of the convex programming problem.

4. Implementation issues and computational results

In this section, we describe an implementation of the cutting plane algorithm presented in the previous section. This implementation has been tried in a PC environment (using Microsoft FORTRAN), on an IBM 3090-200e mainframe (using vs FORTRAN v2r4), and on an HP 9000-720 Apollo workstation with RISC coprocessor (using FORTRAN-77).

We present the results from the workstation, although similar results (in terms of iteration counts and solution accuracies) may be obtained on the other platforms. The exact configuration of the HP 9000-720 workstation which was used is as follows:

(1) Storage: 48 megabytes.

(2) UNIX version 8.05 Operating System.

4.1. ABOUT THE TEST PROBLEMS

Table 1 summarizes the 15 convex programming test problems which were tried. The first 14 problems come from [21, 1]; the 15th problem comes from [17]. All are instances of the geometric programming problem. Problems 13 [5], 14 [6] and 15 [17] come from real applications. The exponential variable transformation ($t_i = e^{y_i}$)

Table 1

Problem definition.

Problem number	Problem in [21]	Number of Vars	Number of Const	Degree of difficulty	Initial point
1	1	2	1	1	Phase 1
2	2	4	2	1	Phase 1
3	3	3	1	5	Phase 1
4	4	4	1	7	Phase 1
5	5	11	3	19	e^T
6	6	4	3	3	Phase 1
7	7a	8	7	3	Phase 1
8	7b	8	7	3	Phase 1
9	8	7	7	40	e^T
10	9a	7	4	10	Phase 1
11	9b	7	4	10	Phase 1
12	9c	7	4	10	Phase 1
13	10	10	7	9	Phase 1
14	11	22	36	50	Phase 1
15	–	30	40	274	Phase 1

is used for each of the problems, thereby eliminating the need for explicitly maintaining the positivity constraint $t_i > 0$. Also the objective functions for each of these problems have been made linear, using the transformation given in section 1. Finally, in section 4.5 we also present some illustrative results for some numerically difficult semi-infinite programming problems [3]. These problems were solved by using uniform discretizations of the index set.

Although two of the problems were small enough to obtain an initial feasible point through inspection, most of them required the use of a phase 1 stage (described below) to find an initial interior point. Problems 1–4, 6–8, 10–14 required the phase 1 stage; for problems 5 and 9, the initial interior point used was e.

4.2. ABOUT THE IMPLEMENTATION

In this subsection, we discuss several of the important implementation techniques we used in developing our path-following cutting plane system. This discussion will center around the main activities performed by the system: generating the search directions, line searching and generating cuts, determining an initial interior point, setting the required parameters, and terminating the algorithm.

4.2.1. Search direction generation

It is well recognized that the computationally costliest step in virtually all interior point methods is the formulation of the normal matrix and the generation of the search direction p (or p_Q). For convenience, we restate the definition of p_Q (4) as the solution of the following linear system.

$$(AS^{-2}A^{\mathrm{T}})_Q p_Q = \frac{b}{\mu} - (AS^{-1})_Q e,$$

where the columns in A and S correspond to the cuts maintained in the index set Q. Frequently, the normal matrices encountered are relatively large and sparse. With such matrices, various techniques are typically used to preserve sparsity and speed-up the computations. However, this is not the case here. The number of variables (and thus the size of $AS^{-2}A^{\mathrm{T}}$) for the problems we consider is relatively small (less than 100).

Also, since the cuts are generated as the gradients of the convex constraints, the normal matrix in general will not tend to be sparse.

Initially, we chose Cholesky factorization to solve the symmetric, positive definite system. For most problems which were tried, Cholesky proved effective in generating search directions of sufficient quality to allow the algorithm to find solutions with duality gaps of 10^{-9} to 10^{-12}. However, for those problems where the condition number of $AS^{-2}A^{\mathrm{T}}$ becomes too large (10^{15} and greater), Cholesky fails to find reliable search directions, which in turn causes our algorithm to fail. To improve the reliability of our algorithm, we switched to QR factorization. This more stable technique allowed the algorithm to solve each of the problems in table 1 from any interior point we tried. Although QR factorization is computationally more costly than Cholesky, we feel that the added stability justified the additional effort.

Currently, we use LINPACK's QR factorization routines DQRDC and DQRSL with the standard column pivoting. We propose several possible enhancements to our

current approach. One possible improvement would be to use the faster Cholesky technique until numeric accuracy becomes a problem. QR could then be used to provide added stability. Another approach would be to investigate more exotic column pivoting schemes to help improve the stability of QR. One possible ordering could be based on the closeness of each column to the current iterate. Another approach would be to replace QR factorization with singular value decomposition. All of these possible improvements are left as areas of future investigation.

The number of cuts which are maintained in the index set Q directly influences the efficiency of the algorithm. Obviously, as this algorithm proceeds this set will tend to grow, which causes extra storage demands to maintain A_Q and slows the formulation of the normal matrix. To control these effects, a column adding–deleting strategy is used.

Initially, A_Q is set to

$$A_Q = [I, -I],$$

where I is the $m \times m$ identity matrix. These "box" constraints are set large enough to contain the original feasible region of (**CP**) and to provide upper and lower bounds for the variables in the model. Our computational experience indicates that the size of this box is not crucial; i.e. a large box around the feasible region is sufficient to start the process. From this starting polytope, the cuts are generated as described in section 3. These cuts are included in the index set Q, which is allowed to grow in cardinality up to some initial maximum value. We chose this initial maximum to be a constant multiple of the variable dimension; i.e. for our system, this maximum was set at $4m$.

Once this maximum was reached, the following adding–deleting procedure is used. Whenever a new cut is to be added, an old cut is identified for removal. The cut which is removed is that cut from the final $3m$ columns of A_Q which has the highest dual slack value s_i. Once this column is found, the new cut simply overwrites the old one, c_i is updated to reflect the new cut, and the algorithm continues. The first m columns of A_Q are left unchanged so as to ensure that A_Q is always of full row rank. The second m columns of the initial box may be overwritten (be generated cuts) as necessary.

Other schemes and other default values could be used to control the maximum number of active cuts. For example, the first $2m$ columns could be retained and the new cuts could overwrite the final $2m$ columns. Theoretically, retaining the first $2m$ columns is unnecessary to ensure that A_Q is of full row rank. We have also found after extensive experimentation that retaining these extra m columns does little to improve the numeric stability of the algorithm. We prefer to maintain $3m$ columns of generated cuts, as opposed to $2m$. Of course, maintaining all of the box constraints or increasing this initial maximum may be preferable on some problem sets. Fine-tuning this choice on other problem sets is left as a topic of further investigation.

There is one potential flaw with this scheme. If the initial maximum size of Q is set too low, then there is the possibility the discarded columns from A_Q may still

contain valuable information and removing them may significantly shift the central path, thus making is difficult to recenter the current iterate. Unfortunately, intelligently choosing the optimal maximum for a problem a priori is difficult. Instead, we propose the following adaptive technique. Initially, a maximum is chosen and maintained as described above. This maximum is kept until the system senses that recentering has become a problem, i.e. until the number of minor iterations required to recenter has exceeded some threshold value. We chose a value of $4m$ for this threshold. When this threshold is reached, the initial maximum is increased by m so that the next m columns generated may be included in A_Q without the removal of any of the previous cuts currently resident in A_Q. Based on the allowed memory allocation for the program, a final maximum is imposed on the cardinality of Q, beyond which the set is not allowed to grow. If difficulties in recentering occur and the final maximum cardinality of Q has been reached, the system is terminated.

For the problems reported below, the initial maximum was set at $4m$. For the problems reported here, there was never any need to increase this maximum. Note if a phase 1 stage was required or if the objective function needed to be linearized, then the number of variables maintained by the program may be as much as two greater than that reported in table 1. These extra variables are included in the $4m$ maximum. Although this adaptive strategy is a slight deviation from the procedure described in section 2, we found it to be practically more efficient. This strategy allows for easier control of memory and for increased speed in calculating search directions.

Finally, it should be noted that our cutting plane algorithm does not ensure monotonicity in the objective values $b^T y$ for the iterates generated. However, for this system we chose to impose objective monotonicity. This was done by adjusting the barrier parameter μ in any iteration where the generated direction would cause the objective value to fall, i.e. if $b^T p_Q < 0$, then μ is adjusted to

$$\mu = \frac{b^T (AS^{-2}A^T)_Q^{-1} b}{e^T (AS^{-1})_Q^T (AS^{-2}A^T)_Q^{-1} b}. \tag{8}$$

A new direction p_Q is recomputed with this updated μ. Note that this approach uses the previous factorization and only requires two additional QR solves.

4.2.2. *Line searching and cut generation*

As stated, the algorithm may require two line searches to be performed. The first is required in the Center-and-Add-Constraints procedure. This line search is used to find the maximum of the logarithmic barrier function along the line segment which starts at the current iterate and extends in the p_Q direction. This is done by finding a root of the derivative of the logarithmic barrier function with respect to α, i.e. find α such that

$$\frac{b^T p_Q}{\mu} - \sum_{i=1}^{n} \frac{\alpha \delta s_i}{s_i - \alpha \delta s_i} = 0,$$

where $\delta s_i = (A_Q^T p_Q)_i$. This line search is performed accurately and quickly by a hybrid bisection and secant method.

Depending on how the cuts are generated, a second line search may be required. As shown in the concretized version of Add-Cut and ORACLE procedures, a line search is needed to find y^b, a point of the boundary of the convex region. We have found this to be a reasonable strategy as long as the gradients of the constraints are easy to compute. This was the case for the geometric programming problems in table 1; closed form gradients are easily calculated for each. However, if the constraints are not differentiable or if the gradients are difficult/expensive to compute, then we recommend basing the cuts on some point which does not require a line search to be performed, such as y or $\tilde{y}$ as shown in (7). In section 4.3, we report results based on cuts generated with line searching (using y^b) and without line searching (using y).

A line search strategy similar to the first is also used here. However, numeric difficulties may arise in this search. In particular, if the norm of the gradient at the point of intersection is quite large, it may be numerically infeasible to find a point $\bar{y}$ on the line segment such that $|f_k(\bar{y})| < \varepsilon$ for some small, positive tolerance ε near machine precision. Instead, it may well happen that two points, y_1 and y_2, on the line segment are found such that

$$\begin{aligned} \|y_1 - y_2\| &< \varepsilon, \\ f_k(y_1) &< -\varepsilon, \\ f_k(y_2) &> \varepsilon. \end{aligned}$$

If this is the case, then it is possible that the cut described in Add-Cut may be invalid (it may remove feasible solutions) and should be corrected as follows. Let $i = \arg\min_{i=1,2}(|f_k(y_i)|)$. Then

$$\begin{aligned} a &= \nabla f_k(y_i), \\ \gamma &= \nabla f_k(y_i)^T y_i - f_k(y_i), \\ \mathcal{P} &= \mathcal{P} \cap \{a^T y \le \gamma\}. \end{aligned}$$

Finally, we recall that the cuts are removed on the basis of the associated dual slack values. It is clear that this is not a scaling independent measure of closeness to the current iterate. To compensate for this, the cuts which are generated are all scaled so that $\|a_i\| = 1$ for all columns i. This allows the dual slacks to be a more reliable measure of closeness.

4.2.3. Finding an initial feasible interior point

The cutting plane method we propose requires an initial interior feasible point (in terms of the original convex region) to start. For some problems, such a point may

be easily found by inspection. However, finding such a point by inspection for larger or more complex problems in general is very difficult.

To solve this problem, we may use either the well-known phase 1–phase 2 or big M approaches, as needed. As with the LP case, the big M method allows the convex programming problem to be solved in one pass. For some choice of the parameter M, the following problem replaces the original problem (**CP**).

$$(\mathbf{CP}_M) \qquad \begin{aligned} &\text{maximize} && b^{\mathrm{T}}y - M\tau \\ &\text{subject to} && f_i(y) - \tau \le 0, \quad 1 \le i \le n, \\ & && \tau \ge 0. \end{aligned}$$

By adjusting τ to be large enough, an initial interior solution (y^0, τ) can easily be chosen. Starting from this point, $(\mathbf{CP}_M)$ is then solved using the algorithm described in section 3. If M is chosen to be large enough, τ will be set to zero in the optimal solution, hence simultaneously solving the original convex programming problem. If during the iterations $y \in \mathcal{F}^0$ occurs, one sets $\tau = 0$ and continues to solve the original problem.

An alternative approach for finding an initial interior iterate is to apply the phase 1–phase 2 approach. Consider the following phase 1 problem.

$$\begin{aligned} &\text{maximize} && -\tau \\ & && f_i(y) - \tau \le 0, \quad 1 \le i \le n. \end{aligned}$$

With phase 1, an artificial variable is added to each of the constraints. This allows an interior feasible point to be easily chosen for this augmented system. With the objective of maximizing the negative artificial variable, the cutting plane method is allowed to iterate on the augmented problem until an interior feasible point is found for the original problem. At this point, phase 1 terminates and the method proceeds on the original problem using the interior point found previously as a starting point.

We chose to implement the phase 1 approach over the big M because of the difficulty of choosing an appropriately large value for M on the various problems. See section 4.3 for a summary of the performance of the phase 1.

4.2.4. Parameter settings

In this subsection, we discuss several of the parameter settings which are required by the algorithm. Our experience has been that the performance of the method (measured in terms of total CPU time and in final solution accuracy) is insensitive to each of the settings μ_0, θ, t_c and $|Q|$. The last parameter, t_a, has a definite impact on the final solution accuracy which can be obtained and is closely tied to machine accuracy.

The initial value of μ, μ_0, is arbitrarily set to 100. The algorithm is insensitive to this setting because this value may be automatically changed, as shown in (8). This change is frequently invoked during the initial iterations when large movements of the iterate occur.

The parameters θ and t_c are set at 0.9 and 10^{-16}, respectively. Taking a small θ (much less than 1/2) would definitely slow down the algorithm, but the algorithm performance seemed insensitive for the values $1/2 < \theta < 0.99$.

Recall, in section 2 t_c is defined as the convergence parameter, i.e. the maximum possible duality gap at the algorithm termination. By setting this parameter to such an optimistically low level (approximately machine accuracy), we allow the algorithm to either find an exact solution or to proceed as far as possible until numeric difficulties cause the algorithm to end prematurely. This topic will be discussed further in section 4.2.5.

Recall that the initial maximum size of Q which is allowed was heuristically set to $4m$. Other values for this parameter were tried, but none seemed to have a clear advantage in the majority of cases tried. We chose this level because it was a reasonable compromise in the tradeoff between memory requirements and computation speed. Recall that for the problems investigated, this maximum was not increased.

The final parameter to be set is found in the Center-and-Add-Constraints procedure. The t_a parameter monitors how closely an iterate is allowed to drift toward the boundary of the convex region before a new cut is added. Using exact arithmetic, this parameter could be ignored and the iterate could be allowed to approach arbitrarily close to the convex boundary, although this would typically result in significantly higher iteration counts. Unfortunately, the computer uses far from exact arithmetic and allowing the iterate to approach the boundary can cause serious numerical problems. In particular, as the iterate draws close to the boundary, the associated slack variable draws correspondingly close to zero. This in turn causes the other slack variables to become less significant in the normal martix calculation $AS^{-2}A^{\mathrm{T}}$, causing this matrix to become seriously ill conditioned. The proper setting of the t_a parameter can help to alleviate these numeric problems. t_a should be set so that if the calculated point has a dual slack which becomes small enough to cause the other slacks to lose significance, a new cut is generated, shifting the central path away from the constraint.

Judging the exact value that t_a should be set at a priori to avoid this problem is difficult, but we do use a heuristic which worked very well in practice. Let $y_k \in \mathcal{F}^0$ be the current iterate and $y = y_k + \alpha p_Q$ be the trial point. The decision rule is as follows:

Find the closest convex constraint i such that:

$(f_i(y) \geq 0.0)$ or

$(f_i(y_k) < -10^{-k_1}$ and $f_i(y) > -10^{-(k_1+k_2)})$ or

$(f_i(y_k) \geq -10^{-k_1}$ and $f_i(y) > -10^{-k_2} f_i(y_k))$

If such a constraint i exists then
  Add a new cut for constraint i and recompute a new trial point y
Else
  No new cut is required – update the iterate.
End If

Note $t_a = 10^{-k_2}$. This rule simultaneously prevents one slack from becoming too small too quickly while allowing the iterate to draw close to the optimal solution towards the end of the procedure. We set k_1 and k_2 so that $2(k_1 + k_2)$ is less than the number of digits of machine accuracy. This sum is multiplied by 2 because the slacks are squared in the normal matrix calculation. For our implementation, we chose k_1 and k_2 to be 6 and 2, respectively; thus, $t_a = 10^{-2}$.

4.2.5. The termination of the system

As mentioned in the previous section, one method of terminating this procedure is when the duality gap falls near to machine tolerance. Unfortunately, this is not usually obtainable in practice. Typically, the numeric difficulties discussed above will cause the QR procedure to generate unreliable search directions as the iterate draws near the optimal solution. The algorithm is terminated when the error in the generated direction (as measured by the norm of the residual vector) grows beyond a fixed tolerance. Such a termination technique is more realistic.

It should also be pointed out that if the problem is stated with finite precision or if the constraint function/gradients are difficult to compute accurately, then there is little sense to compute a solution vector with greater accuracy than the original data. The algorithm should be terminated when the duality gap reaches this accuracy level.

4.3. COMPUTATIONAL RESULTS

In this section, we present the computational results we found with our cutting plane algorithm on the problem data set previously described. Our results are summarized in tables 2, 3, and 4. Table 2a summarizes our findings using the feasible boundary technique of generating cuts (with line searching); table 2b summarizes our findings using a y-based cut generation technique (without line searching). All other parameters and settings were identical for both sets of runs. We feel these results are very comparable. It appears that the gain which is received from not requiring the extra line search is approximately lost due to the need for addition cuts. Tables 3, 4, and 5 are based on results from table 2a.

There are several points which we wish to highlight from tables 2a and 2b. First, there seems to be a remarkable consistency in the major iteration counts listed. This consistency is a direct result from the strategy we use to reduce the logarithmic

Table 2a

Problem results overview with line search.

Problem number	Max Q size	Major iters.	Number of Matrix factor.	Cuts	Fnct evals. 1000's	Duality gap	Total CPU (s)
1	16	17	53	27	1.3	9.9e – 15	0.23
2	24	19	66	27	1.9	2.6e – 15	0.47
3	20	18	78	48	2.2	4.6e – 12	0.47
4	24	19	108	74	4.1	1.0e – 09	0.99
5	48	12	138	99	3.9	1.0e – 10	3.56
6	24	19	94	52	2.9	4.8e – 09	0.83
7	40	19	118	65	4.8	7.2e – 13	2.17
8	40	18	117	79	4.6	7.2e – 13	2.08
9	32	13	150	115	6.2	6.4e – 13	2.01
10	36	17	181	128	6.1	9.7e – 11	3.25
11	36	17	168	114	5.6	4.7e – 10	2.51
12	36	17	158	108	5.5	1.5e – 10	2.36
13	48	20	207	135	9.9	8.1e – 07	6.54
14	96	17	327	253	32.4	2.5e – 11	39.60
15	128	21	557	471	74.3	1.4e – 09	162.88

Table 2b

Problem results overview without line search.

Problem number	Max Q size	Major iters.	Number of Matrix factor.	Cuts	Fnct evals. 1000's	Duality gap	Total CPU (s)
1	16	18	61	34	0.5	2.4e – 15	0.28
2	24	19	62	23	0.9	4.3e – 15	0.43
3	20	19	84	50	0.9	5.6e – 12	0.50
4	24	19	100	68	0.5	1.4e – 10	0.83
5	48	14	139	106	1.3	4.1e – 12	3.65
6	24	19	113	70	1.0	1.5e – 08	1.12
7	40	20	166	108	2.8	5.6e – 14	3.08
8	40	18	166	112	2.8	2.6e – 13	2.94
9	32	14	152	123	2.7	1.6e – 12	2.00
10	45	18	233	190	2.7	6.3e – 11	3.96
11	36	18	189	138	2.1	2.7e – 11	2.84
12	36	20	182	128	2.0	2.7e – 12	2.76
13	48	22	259	176	4.5	3.3e – 07	8.13
14	96	19	321	262	24.3	2.6e – 11	38.60
15	128	24	652	560	54.6	4.7e – 10	195.67

Table 3

Phase 1–phase 2 breakdown.

Problem number	Phase 1 Major iters.	Phase 1 Normal factor	Phase 1 CPU (s)	Phase 2 Major iters.	Phase 2 Normal factor	Phase 2 CPU (s)
1	2	8	0.03	15	45	0.20
2	2	9	0.06	17	57	0.41
3	4	9	0.07	14	69	0.40
4	3	10	0.09	16	98	0.90
6	4	25	0.20	15	69	0.63
7	4	29	0.50	15	89	1.67
8	3	21	0.33	15	96	1.75
10	4	30	0.45	13	151	2.80
11	4	30	0.43	13	138	2.08
12	4	30	0.45	13	128	1.91
13	6	44	1.21	14	163	5.33
14	4	41	3.71	13	286	35.89
15	4	119	30.19	17	438	132.69

barrier parameter μ. At the start of each major iteration, this parameter is reduced by a factor of 10. Since the algorithm theoretically stops when the duality gap is less than t_c and on the central path the gap is the product of $n\mu$, then this consistency should not be surprising. A more reliable measure of the work required to solve the problem may be found in the number of normal matrix formulations and factorizations, and in the number of cuts generated to obtain a given level of solution accuracy.

Also, in tables 2a and 2b we see that duality gaps of 10^{-10} to 10^{-15} can typically be achieved with this method. Although machine accuracy (10^{16}) is not obtained, the method presented here typically does significantly better than the previous methods proposed in [21] and [1] to solve these problems in terms of solution accuracy. Our results are comparable and frequently better than Vial's [28] results.

Note that for problem 13, our method had difficulties obtaining the usual level of precision; a gap of only 8.1e – 7 was possible. In this problem, there was an enormous range on the coefficients which exacerbated the numeric difficulties addressed earlier; i.e. the range caused the iterate to fall too near a constraint before a higher degree of precision could be obtained.

Table 3 highlights the phase 1–phase 2 comparison on the problems which required a phase 1 stage to be performed. In general, we found that approximately 15–25% of the total computational effort is spent in the phase 1 stage.

Table 4 presents a percentage CPU breakdown for the largest problem that was solved. As might be expected, the majority of effort of this method is spent formulating the normal matrix and computing the search direction. In this system, the normal

Table 4

Time percentage breakdown for problem 15.

Normal matrix formulation/solving	77%
Evaluating convex constraints/gradients	7%
Line search operations	7%
Pricing ($A^T p_Q$ calculations)	5%
Miscellaneous	4%

matrix is reformulated and refactored at each iteration. It is possible that numerically this is not really required. It is possible that the normal matrix (and its factorization) could be maintained for several inner iterations without seriously affecting the directions that are generated. This possibly could save a significant amount of effort. Such investigations are left as an area of future research.

4.4. COMPARISON TO PAST WORK

In tables 5a and 5b, a comparison of our results and those of [21, 17, 1, 28] is given. Our method seems more stable on this problem set. The logarithmic barrier path following method is on average several more digits of accuracy in the solution than that of [21] and [1]. Note that the higher degree of accuracy we obtained required only slightly more effort than required in [21] and [1]. Higher degrees of precision are important with these problems because there is the real possibility that the central path may fall close to one of the constraints. If this happens and the method terminates prematurely, then the point which is found to be "optimal" by the method may in fact be far from the true optimum, however the true duality gap is always bounded by the gap presented.

4.5. SEMI-INFINITE PROGRAMMING RESULTS

Finally, as a means of demonstrating the usefulness of this technique for finding the optimal solution to semi-infinite programming problems, we consider the following problem which was extensively tested in [3].

$$\text{Minimize} \quad \sum_{i=1}^{n} x_i / i$$

$$\text{subject to} \quad \sum_{i=1}^{n} s^{i-1} x_i \geq \tan(s) \quad \text{for all } s \in [0,1].$$

As mentioned in theorem 3, we can find an optimal solution to this problem by imposing a uniform grid for s over $[0,1]$; i.e. we solve the generated linear programming problem

Table 5a

Result comparison.

Problem number	Solution accuracy[3]		
	Current	Method [21]	Method [28]
1	9.9e – 15	2.0e – 9	1.6e – 7
2	2.6e – 15	4.0e – 6	2.2e – 9
3	2.6e – 12	3.0e – 5	1.1e – 3
4	1.0e – 9	4.0e – 6	7.3e – 2
5	1.0e – 10	1.1e – 4	1.1e – 6
6	4.8e – 9	4.0e – 5	2.1e – 3
7	7.2e – 13	8.0e – 5	7.5e – 8
8	7.2e – 13	8.0e – 8	6.3e – 6
9	6.4e – 13	7.0e – 5	7.1e – 6
10	9.7e – 11	7.0e – 6	1.2e – 3
11	4.7e – 10	7.0e – 7	1.1e – 4
12	1.5e – 10	7.0e – 5	1.2e – 4
13	8.1e – 7	1.0e – 3	4.2e – 3
14	2.5e – 11	2.2e – 4	6.8e – 10
15	1.4e – 9	**[4]	6.4e – 10

Table 5b

Result comparison to [1].

Problem number	Current method		Method [1]	
	Accuracy	No. of cuts	Accuracy	No. of cuts
1	9.9e – 15	27	4.0e – 7	14
2	2.6e – 15	27	9.0e – 7	35
3	4.6e – 12	48	8.0e – 7	31
4	1.0e – 9	74	9.0e – 7	41
5	1.0e – 10	99	9.0e – 7	84
6	4.8e – 9	52	6.0e – 7	41
7	7,2e – 13	65	7.0e – 7	73
8	7.2e – 13	70	9.0e – 7	75
9	6.4e – 13	115	9.0e – 7	73
10	9.7e – 11	128	7.0e – 7	71
11	4.7e – 10	114	8.0e – 7	75
12	1.5e – 10	108	9.0e – 7	89
13	8.1e – 7	135	8.0e – 7	146
14	2.5e – 11	253	9.0e – 7	123
15	1.4e – 9	471		

[3] For the current method and for [21], the value of the barrier parameter μ is reported. This value is multiplied by n to obtain an upper bound for the duality gap. Some of the results of [21] are from the technical report, since they were removed from the final version. Vial [28] reports the achieved relative duality gap. Since this table presents the absolute duality gaps, the reported (relative) duality gaps are adjusted appropriately.

[4] This problem is not solved in [21], but the authors of paper [21] did solve this problem in [17]. The solution presented in [17] has a small typographical flaw; as stated, constraint no. 34 is violated ($0.3125Q_D \leq 1$ where the reported value for $Q_D = 5.38$).

Table 6

Semi-infinite programming results.

Number of vars. (n)	Grid fineness (m)	Major iters.	Matrix factor.	Cuts	Gap	CPU (s)
10	1e2	15	63	24	3.6e – 14	0.6
10	1e3	13	79	35	1.8e – 13	2.1
10	1e4	12	78	39	2.1e – 12	14.1
10	1e5	13	83	42	2.2e – 13	137.4
10	1e6	13	84	42	2.6e – 13	1399.3
20	1e2	9	54	23	1.2e – 8	1.1
20	1e3	10	76	38	6.4e – 10	3.8
20	1e4	9	66	29	3.5e – 8	23.5
20	1e5	10	52	19	8.4e – 10	179.3
20	1e6	10	67	30	9.2e – 10	2310.0
30	1e2	9	48	23	9.0e – 8	1.8
30	1e3	11	59	25	5.9e – 10	4.6
30	1e4	10	51	22	6.0e – 9	25.7
30	1e5	10	52	19	5.2e – 9	244.0
30	1e6	9	55	26	2.0e – 7	2559.2

with m constraints of the form $\sum_{i=1}^{n} s^{i-1}x_i \geq \tan(s)$ for fixed $s = \{0, 1/m, 2/m, \ldots, 1\}$. Using a variety of finenesses, we have solved this semi-infinite programming problem for $n = 10, 20, 30$. The results are summarized in table 6.

Note that, as a finite subset of the infinitely many linear constraints is used in the discretization, we cannot be sure without minimizing the one-dimensional nonlinear nonconvex function $\min \psi(s) = \sum_{i=1}^{n} s^{i-1}x_i^* - \tan(s)$ if the obtained optimal solution x^* is really feasible for the semi-infinite problem or not. However, the dual optimal solution is feasible for the dual semi-infinite programming problem. To keep the presentation compact, this extra one-dimensional minimization is not discussed here.

5. Conclusions

As reported in the literature, the analytic central cutting plane methods are more stable than other cutting plane methods [1,9]. We also presented a new logarithmic barrier cutting plane method. In this method, the localization is done interactively (more drastically) without getting too close or calculating explicitly the "center" of the actual localization.

Proving global convergence (possibly polynomial complexity) of our logarithmic barrier cutting plane method remains an open problem, although we have seen that, based

on the discretization of the feasible set, polynomial complexity (polynomial in the number of discretization points) can be derived based on the analysis of [13, 15, 16].

We have also demonstrated that the logarithmic barrier cutting plane method is a computationally viable technique for solving several geometric and semi-infinite programming problems. We have found this method to be numerically more stable than some previous methods with comparable amounts of computational effort. However, some of the methods produce both primal and dual optimal solutions.

In terms of implementation issues, we leave several questions open as areas of possible future research. The first is whether a better adaptive heuristic can be found for directing the reduction of the logarithmic barrier parameter μ, which would reduce the total number of factorizations required. The second is developing strategies for maintaining the normal matrix for consecutive inner iteration without seriously eroding the quality of search directions that are generated. For larger problems with difficult to evaluate functions, additional methods for generating cuts should be further explored. Finally, various techniques for improving the stability of generating the search direction should also be investigated.

Acknowledgement

The authors are grateful to both of the anonymous referees for the constructive remarks that improved the quality of the paper significantly.

References

[1] O. Bahn, J.L. Goffin, J.-Ph. Vial and O. Du Merle, Implementation and behavior of an interior point cutting plane algorithm for convex programming: An application to geometric programming, Discr. Appl. Math. 49(1994)3–23.

[2] E.W. Cheney and A.A. Goldstein, Newton's method for convex programming and Tchebycheff approximation, Numer. Math. 1(1959)243–268.

[3] I.D. Coope and G.A. Watson, A projected Lagrangian algorithm for semi-infinite programming, Math. Progr. 32(1985)337–356.

[4] J. Elzinga and T.G. Moore, A central cutting plane algorithm for the convex programming problem, Math. Progr. 8(1975)134–145.

[5] A.V. Fiacco and A. Ghaemi, A sensitivity analysis of a nonlinear water pollution control problem using an upper Hudson River database, The George Washington University, Washington, DC (1982).

[6] A.V. Fiacco and A. Ghaemi, A sensitivity and parametric bound analysis of an electric power GP model: Optimal steam turbine exhaust annulus and condenser sizes, serial T-437, The George Washington University, Washington, DC (1981).

[7] J.L. Goffin, A. Haurie and J.-Ph. Vial, Decomposition and nondifferentiable optimization with the projective algorithm, Manag. Sci. 38(1992)284–302.

[8] P.R. Gribik, A central-cutting-plane algorithm for semi-infinite programming problems, in: *Semi-Infinite Programming*, ed. R. Hettich, Lecture Notes in Control and Information Sciences Vol. 15 (Springer, New York, 1979) pp. 66–82.

[9] J.L. Goffin and J.P. Vial, Cutting planes and column generation techniques with the projective algorithm, J. Optim. Theory Appl. 65(1988)409–429.

[10] P.R. Gribik and D.N. Lee, A comparison of two central cutting plane algorithms for prototype geometric programming problems, in: *Methods of Operations Research* 31, ed. W. Oettli and S. Steffens (Verlag Anton/Hain/Mannheim, Germany, 1978) pp. 275–287.
[11] S.-A. Gustafson, On numerical analysis in semi-infinite programming, in: *Semi-Infinite Programming*, ed. R. Hettich, Lecture Notes in Control and Information Sciences Vol. 15 (Springer, New York, 1979) pp. 51–65.
[12] R. Hettich and K.O. Kortanek, Semi-infinite programming: Theory, methods, and applications, Working Paper, FB IV, Universität Trier, Germany (1991).
[13] D. den Hertog, *Interior Point Approach to Linear, Quadratic and Convex Programming – Algorithms and Complexity* (Kluwer Academic, Dordrecht, The Netherlands, 1994).
[14] D. den Hertog, C. Roos and J.-Ph. Vial, A complexity reduction for the long-step path-following algorithm for linear programming, SIAM J. Optim. 2(1992)71–87.
[15] D. den Hertog, C. Roos and T. Terlaky, A Build-Up variant of the path-following method for LP, Oper. Res. Lett. 12(1992)181–186.
[16] D. den Hertog, C. Roos and T. Terlaky, Adding and deleting constraints in the logarithmic barrier method for linear programming problems, in: *Advances in Optimization and Approximation*, ed. D. Du and J. Sun (Kluwer Academic, Dordrecht, The Netherlands, 1994) pp. 166–185.
[17] S. Jha, K.O. Kortanek and H. No, Lotsizing and setup time reduction under stochastic demand: A geometric programming approach, Working Paper No. 88-12, College of Business Administration, The University of Iowa, Iowa City, IA (1988).
[18] J. Kaliski and Y. Ye, A decomposition variant of the potential reduction algorithm for linear programming, Manag. Sci. 39(1993)757–776.
[19] J.E. Kelley, Jr., The cutting-plane method for solving convex programs, J. Soc. Ind. Appl. Math. 8(1960)703–712.
[20] K.O. Kortanek, Semi-infinite programming duality for order restricted statistical inference models, Working Paper No. 91-18, College of Business Administration, The University of Iowa, Iowa City, IA (1991).
[21] K.O. Kortanek and H. No, A second order affine scaling algorithm for the geometric programming dual with logarithmic barrier, Optimization 23(1990)303–322. Earlier version: Working Paper No. 90-07, College of Business Administration, The University of Iowa, Iowa City, IA (1990).
[22] K.O. Kortanek and H. No, A central cutting plane algorithm for convex semi-infinite programming problems, SIAM J. Optim. 3(1993)901–918.
[23] N. Megiddo, Pathways to the optimal set in linear programming, in: *Progress in Linear Programming, Interior and Related Methods*, ed. N. Megiddo (Springer, New York, 1989) pp. 131–158.
[24] J.E. Mitchell and M.J. Todd, Solving combinatorial optimization problems using Karmarkar's algorithm, Math. Progr. 56(1992)245–284.
[25] J. Renegar, A polynomial time algorithm, based on Newton's method, for linear programming, Math. Progr. 40(1988)59–93.
[26] C. Roos and J.-Ph. Vial, A polynomial method of approximate centers for linear programming, Math. Progr. 54(1992)295–305.
[27] Gy. Sonnevend, An "analytic centre" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming, in: *Lecture Notes in Control and Information Sciences* Vol. 84 (Springer, New York, 1985) pp. 866–876.
[28] J.-Ph. Vial, Computational experience with a primal-dual interior point method for smooth convex programming, Report April 1993, Département d'Economie Commerciale et Industrielle, Université Genève, Genève, Switzerland (1993), to appear in Optim. Meth. and Softwares.
[29] Y. Ye, A potential reduction algorithm allowing column generation, SIAM J. Optim. 2(1992) 7–20.